LOGIC PROGRAMMING & PROLOG

- Concepts of Logic Programming
- Introducing Prolog - Basic Concepts/Usage
- Advanced Data Structures
- Logic Programming Techniques
- Flow of Control
- More Advanced Features
- Concluding Remarks - Looking Ahead

HISTORICALLY

- Developed in 1972 - Still one of a kind
- Influenced by developments in both
  - LISP & ALCOL 68
- Wide range of applications
  - Algorithm Specif.
  - Database Storage/Retrieval
  - Compilers/Language Translators
  - Expert Systems
  - Knowledge-Based Sys.
CONCEPTS OF LOGIC PROGRAMMING

• LOGIC PROG. OFTEN CALLED PROG. WITH RELATIONS

• CONCEPT OF RELATION SIMILAR TO DB CONCEPT.

DEF: A RELATION IS A TABLE WITH A > 0 COLS & AN INFINITE SET OF ROWS. EACH ROW IS CALLED A TUPLE.

• RELATIONS USED TO REPRESENT FACTS AND RULES.

• A FACT IS A STATEMENT OF INFO. THAT MUST BE TRUE

• A RULE IS AN ABSTRACT REPR. OF A RELATION THAT IS USED TO DEDUCE INFO. FROM FACTS.
WHAT IS LOGIC PROGRAMMING?

- Logic Programming
- The use of facts & rules to represent info
- The use of deduction to answer queries.
- When provided with concrete facts & deducing rules, the system can dynamically construct new facts (i.e., deduce), based on:
  - Correct facts
  - Rules for manip. facts
  - Queries using rules/facts.

- How is this done?
INTRODUCTION TO PROLOG

- SPECIFY FACTS, RULES, QUERIES THAT CONSIST OF TERMS

- A TERM IS
  - A NUMBER, i.e., 0, 1972
  - A VARIABLE, i.e., X, SOWCA, STARTS WITH A CAPITAL LETTER.
  - AN ATOM, i.e., LISP, ALGO160, WHICH IS AN ENTITY THAT REPRESENTS ITSELF

- A COMPOUNDED TERM IS
  <ATOM> (<SUBTERM-LIST>)
  - FOCUS OR ARGUMENTS

EXAMPLES

ink (blue, c) \( \gamma \) WILL BE USED TO REPRESENT FACTS & RULES & QUERIES

SYNTAX OF FACTS, RULES, QUERIES FOR EDINBURGH PROLOG:

<fact> :: = <term>.
<rule> :: = <term> :- <term>.
<query> :: = <term>.
<term> :: = <number> | <atom> | <variable> | <atom> (<term>)
<variable> :: = <term> | <number>, <term>

- LET'S BEGIN WITH STANDARD EXAMPLES

- RED IS A PROLOG
  BLUE IS A VEN
A FIRST EXAMPLE

Define parent relation of facts as:

- parent(john, jack).
- parent(john, jill).
- parent(peter, john).
- parent(vicky, steve).
- parent(vicky, sandy).
- parent(harry, anna).
- parent(jim, justin).
- parent(sandi, patric).

Where

parent(x, y) means that x is parent of y.

Place facts in a Unix file, parent_info.

Now, let's consider a Prolog session:

```prolog
gprolog > consult(parent_info).
parent_info consulted ... /
```

```
?- parent(vicky, steve), parent(john, jill). ; 
```

```
; parent(vicky, steve), parent(john, jill).
; "j" - NOT?
```

```
?- parent(steve, harry).
```

```
- YES.
```

```
?- parent(vicky, l), parent(l, m).
```

```
- Query?
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
A second example - Links between prog. langs

* Construct a file called links.info that contains

  link (father, algol 60).
  link (algol 60, cpl).
  link (cpl, bcpl).
  link (bcpl, c).
  link (c, cplus plus).
  link (algol 60, simula 67).
  link (simula 67, cplus plus).
  link (simula 67, smalltalk 80).
  path (l, l).
  path (l, m) = link (l, k), path (k, m).

Which are historical influences links between prog. languages

* What does query

  parent (vick70, l), parent (l, m).

  Ask for?

* Define a grand parent relation with rules

  ?- grand parent (GP, GC) :- parent (GP, P), parent (P, GC).

  How is this used?

  ?- grand parent (vick70, justin).
     yes

  ?- grand parent (x, justin).
     x = vick70

  ?- grand parent (vick70, y).
     y = justin
USE IN A SIMILAR WAY AS FIRST EXAMPLE

? - consult(linked-1to0).
   Yes

? - link (rep1, rep2), link (rep2, c).
   Yes

? - link (rep1, c), link (c, rep2).
   Are there L and M such that link(algo1, L) & link(L, M)?
   L = scale 67
   M = scale 67
   <- if card topped, instead, PROLOG responds
   yes
   no

? - link (rep1, c).
   L = c

? - link (c, rep1).
   L = c

? - link (c, c).
   <- return null facts?

HOW DOES PROL ROLE WORK?

1. path (L, N).
   yes

2. path (L, M) :- link (L, X), path (X, M).
   <- it's recursive

   line 1 - termination condition.

   - recall rules are relations &
   this relation has two tuples

   line 2 - recursively search for lambda LI:

   example

? - path (father, smalltalk 80).
   yes

? - path (scheme, lisp).
   no

? - path (Y, cpl).
   what happens here?

? - path (cpl, Y).
OTHER CONCEPTS

QUESTION: ARE THERE TWO LINKS L & M WITH LINKS TO SAME LANGUAGE N?

?- link(L,N), link(M,N), link(L,M).
L = fortran
N = algol60
M = julan.

WHAT'S WRONG HERE?

?- link(L,N), link(M,N), not(L=M).
L = c
N = cplusplus
M = smalltalk67

NOTE: ORDER IS IMPORTANT!!

?- not(L=M), link(L,N), link(M,N).
no
UNIFICATION IN PROLOG

The process of "matching" PROLOG structures

For example,

?- f(x, b) = f(a, y).
    x = a
    y = b

Since \( f(a, b) \) is an instance of \( f(x, b) \)
& \( f(a, b) \) is an instance of \( f(a, y) \)

Another example

?- f(a, x) = f(y, x).
    x = 0  y = 0 ; machine generated variable
    y = a

?- f(a, x, z) = f(c, y, x, z).
    x = 0
    z = 1
    y = a

ARITHMETIC OPERATIONS

Also supported in PROLOG

"is" - assignment operator
- supports infix expressions

Examples

?- X is 2+3, Y is 5*X.
   X = 5
   Y = 25

?- X is 2+3, X = 5.
   X = 5

?- X is 2+3, X = 2+3.
   no - why?

?- Y is 5*X, X is 2+3.
   Error - why?
BACKTRACKING CONCEPTS

parent (vick, ash).
parent (john, paul).
parent (harry, paul).
parent (jill, peter).
parent (peter, paul).

BACKTRACKING WITH PLACE-HOLDERS

parent (F, C), parent (M, C), not (F=M).

HOW ARE SUBOBSERVATIONS PROCESSED?

ANOTHER EXAMPLE

pop (usa, 205).
pop (india, 540).
pop (china, 800).
pop (brazil, 08).
area (usa, 3).
area (india, 1).
area (china, 4).
area (brazil, 2).

POPPATIONS IN MILLIONS

AREA IN MILLIONS

OF SQ. MILES

RULE FOR POPULATION DENSITY:

density (X, Y) :- pop (X, P),
area (X, A),
Y is P/A.

density (usa, Y).
density (china, Y).

GTC...
ADVANCED DATA STRUCTURES

- STRUCTURES - PROLOG EQUIV. TO RECORDS
  - "NESTED" FACTS
- LISTS
- TREES
- A HELPFUL RULE FOR USING PROLOG

- BUILT IN PROLOG PREDICATES

- BASICALLY PROVIDE A MORE COMPLETE OVERVIEW OF PROLOG THAN OUR TEXTBOOK -

Structures in Prolog

- REPRESENTED REAL-WORLD OBJECTS
  - KEEP TRACK OF BOOKS/TITLES/AUTHORS

book(ringworld, author(larry, niven)).
book(presumed_innocent, author(scott, tradow)).
book(macroscope, author(piers, anthony)).
book(bourne_identity, author(robert, ludlum)).

Pictorially:

```
book
  bourne_identity
    author
      robert
      ludlum
```
LISTS IN PROLOG

- MANY DIFFERENT FORMS

- [a, b, c]
- []
- [X|Y] WHERE Y IS AN INITIAL SEQUENCE & Y IS A TRAILING LIST OF ELEMENTS.
- [a, b, c, []]
- [a, b|c]
- [a|b, c]

- UTILIZE UNIFICATION TO DECOMPOSE LIST

?- [H|T] = [a, b, c].
  H = a
  T = [b, c]

?- [a|T] = [H, b, c].
  T = [b, c]
  H = a

NOW, SOME BASIC LIST FUNCTIONS

- append FOR LISTS

  append([H|X], Y, [H|Z]) :- append(X, Y, Z).

  append([], Y, Y).

THE RESULT OF APPENDING THE EMPTY LIST & Y IS Y.


?- append([a,b], [c,d], Z).
  Z = [a,b, c,d].

?- append([a,b], Y, [a,b,c,d]).
  Y = [c, d]

?- append(X, [c,d], [a,b,c,d]).
  X = [a,b]

?- append(X, [d,c], [a,b,c,d]).
  X = [a,b]

?- append(X, Y, [a,b,c,d]).
  NO

WHAT HAPPENS?
MEMBER FOR LISTS

- member(M, [M|_]).
- member(M, [T|_]) :- member(M, T).

M is a member of a list with head M.
M is a member of a list if M is a member of its tail.

"_" is match anything/don't care

Rules are checked in order during query, that's why rules are ordered as given.

EXAMPLES
?- member(a, [1,2,c,d]).
   no
?- member(b, [1,2,c,d]).
   no
?- member(c, [1,2,c,d]).
   yes

BINARY SEARCH TREES

RECALL ML

data-type tree = empty | node of int * (tree, tree)

IN PROLOG

empty - ATOM FOR EMPTY TREE/SUBTREE

node (k, S, T) - TREE

To construct trees, use

X is node(3, node(4, empty, empty),
      node(5, node(6, empty, empty),
          node(7, empty, empty))

X:

8-23
A MEMBER FUNCTION FOR TREES

\*member \((k, \text{node}(N,S,T))\).
\*member \((k, \text{node}(N,S,-))\) :- \(k < N\), member\((k,S)\).
\*member \((k, \text{node}(N,-,T))\) :- \(k > N\), member\((k,T)\).

"-" : PLACE HOLDER FOR UNNAMED VALUE

\*REWRITE THIS RULE

\*member \((k, U)\) :- \(U = \text{node}(N,S,T)\), \(k = N\).

WHAT ABOUT

insert : insert \(k\) into \(S\) to get \(T\)
delete : remove an integer from tree

---

A Useful vi Rule

- This routine throws you into "vi filename". When you exit vi, it reconsults the file, bringing in any changes you made.

\[\text{vi(Filename)} :-\]
\[\text{name(Filename, Filestring)},\]
\[\text{append("vi", Filestring, Command)},\]
\[\text{system(Command)},\]
\[\text{reconsult(Filename)}.\]

\[\text{append([], List, List)}.\]
\[\text{append([Head \mid List1], List2, [Head \mid List3])} :-\]
\[\text{append(List1, List2, List3)}.\]

---

PLACE IN A FILE & CONSULT WHEN YOU START PROLOG.

THEO YOU CAN ENTER/LEAVE VI WITHOUT LEAVING PROLOG.
**Built-in Prolog Predicates**

- Adding facts/rules using `assert`
  
  ```prolog
  ?- assert ( links (lisp, scheme)).
  ?- assert ( [rule(X,Y)]).  % OUR EXTRA PARES FOR RULES
  ?- assert ( [rule(Y,Y) ; .... ]).
  ```

- Show all clauses
  
  ```prolog
  ?- listing (links).
  TRY THESE OUT!!
  ?- listing (paths).
  ```

**Logic Programming Techniques**

- Two concepts dominate Prolog
  - **Backtracking** - allow a SLD to be found if one exists
    - thru an exhaustive search
  - **Unification** - allow variables to be used as placeholders for data to be filled in later.

- **First Technique - Guess & Verify**
  - Queries of the form:  
    - Is there an S such that
    - `subgoals{ guess(S) & verify(S) }`?
    - continue guessing until a SLD is found

8-27
DETOUR TO BACKTRACKING CONCEPTS

- Four Queens Problem - Can you place 4 queens on a 4x4 chess board so that no two attack one another?

Backtracking to a solution.

Queens movements.

1. Try to move Q on row 3 - can't
2. Try to move Q on row 2 - no more possible -
3. Move Q on row 1
4. Start over!!

CONCEPTUALLY, THIS IS HOW BT WORKS IN PROLOG!!

8-29

GUESS & VERIFY TECHNIQUE

- A FIRST EXAMPLE

\text{overlap}(x, y) : - \text{member}(m, x), \text{member}(m, y).

- Two lists overlap if they have a common element \( m \).

- \text{member}(m, x) - GUESS AN \( m \) FROM LIST
- \text{member}(m, y) - VERIFY \( m \) IS IN LIST \( y \)

- HOW DOES THIS ACTUALLY WORK IN PROLOG?

- LET'S TRACE THROUGH AN EXAMPLE
RECALL MEMBER

\[ \text{member}(M, [a, b, c, d]). \]
\[ \text{member}(M, [e, f]). = \text{member}(M, T). \]

TRACE EXECUTION OF

\[ \text{overlap}([a, b, c, d], [1, z, e, f]) \]
\[ \Rightarrow \text{equiv. to} \]
\[ \text{member}(M, [a, b, c, d]), \text{member}(M, [1, z, e, f]). \]

FIRST SUBGOAL \[ \rightarrow M = a \]
CHECK 2ND SUBGOAL \[ \rightarrow \text{false} \]
BACK TO 1ST SUBGOAL \[ \rightarrow M = b \]
CHECK 2ND SUBGOAL \[ \rightarrow \text{false} \]
BACK TO 1ST SUBGOAL \[ \rightarrow M = c \]
CHECK 2ND SUBGOAL \[ \rightarrow \text{true} \]

(Note: ORDER AFFECTS PROLOG EFFICIENCY)

\[ \bullet \] CHOOSE SUBGOAL WITH FEWEST POSSIBLE SOLNS AS "GUESS" SUBGOAL.

WHAT HAPPENS WHEN GOAL ORDER IS NOT CAREFULLY CHOSEN?

CONSIDER:

\[ ?- X = [1, z, 3], \text{member}(a, X). \]

\[ \text{no} \]

VERSUS

\[ ?- \text{member}(a, X), X = [1, z, 3]. \]

[INFINITE COMPUTATION]

GUESS GOAL member(a, X) HAS AN INFINITE NUMBER OF SOLUTIONS, NAMELY

\[ X = [a, \_]; \ a = 1\text{ST EL. OF } X \]
\[ X = [\_ , a, \_ ]; \ a = 2\text{ND EL. OF } X \]
\[ X = [\_ , \_ , a, \_ ]; \ a = 3\text{RD EL. OF } X \]
\[ \vdots \]

\[ \text{ETC.} \]
**Flow of Control**

- **Informally,**
  
  Algorithm = Logic + Control,

  Where

  Logic refers to Rules, Facts, & Queries in a Prolog Program

  Control refers to how Logic
  interprets the Rules/Facts
  to compute a response to
  a Query.

- **Control in Prolog, determined by**

  1. **Goal Order** - choose the leftmost subgoal

  2. **Rule Order** - select 1st applicable rule

  **Recall,**

  Goal order = Left to right eval
  Rule order = Rules from top-to-bottom
  as loaded into DB.

**Overall** - When determining a response to a query, both
- Goal order within a query
- Rule order of rules/facts in DB

play a critical role.

**Control is Prolog** - how Logic Evaluates Queries

Start with a query as current goal
while the current goal is nonempty do
  choose the leftmost subgoal;
  if a rule applies to a subgoal then
  select the first applicable rule;
  from a new current goal;

  else
    backtrack;

end while

Succeeded
LET'S EXAMINE HOW PROLOG WORKS
USING A COLLECTION OF LISP RULES

append (C, Y, Y).
append ([H1X], Y, [H1Z]) :- append (X, Y, Z).

prefix (Y, Z) :- append (X, Y, Z).

\[ \text{IS THERE SOME } Y \text{ SUCH THAT} \]
\[ X \text{ APPEND } Y = Z? \text{ IF SO,} \]
\[ \text{THEN } X \text{ IS A PREFIX OF } Z. \]

suffix (Y, Z) :- append (X, Y, Z).

\[ \text{SIMILAR TO PREFIX} \]

append2 ([H1X], Y, [H1Z]) :- append2 (X, Y, Z).
append2 ([], Y, Y).

A 2ND, INTERESTING
VERSION OF APPEND, WITH
THE 700 RULES REVERSED
IN DB.

EFFECT OF GOAL ORDER ON QUERY EVAL.

\[ \text{IS } S \text{ A SUBSET OF } Z. \]

\[ X \quad \leftarrow \quad \]
\[ \quad \rightarrow \quad \]
\[ \quad \rightarrow \quad \]
\[ \quad \rightarrow \quad \]
\[ X \text{ IS A PREFIX OF } Z \& \]
\[ S \text{ IS A SUFFIX OF } X. \]

\[ \text{ALTERNATE GOAL ORDERS IN PROLOG.} \]

\[ ? - \text{prefix}(X, [a,b,c]), \text{suffix}([c], X). (\text{false}) \]
\[ \text{no} \]

\[ ? - \text{suffix}([c], X), \text{prefix}([X, [a,b,c]]). \]
\[ \text{infinite computation} \]
**EFFECT OF RULE ORDER ON A QUERY**

?- append(X, [C], Z). \(\exists Y \) such that \(X|\{C\}\) is \(Y\)?

\[
\begin{align*}
X &= [C] \\
Z &= [C] \\
Y &= [-1] \\
Z &= [-1, C] \\
Y &= [-1, -2] \\
Z &= [-1, -2, C]
\end{align*}
\]

No more solutions

?- append2(X, [C], Z). \(\exists Y \) why does \(\text{infinit computation}\) \(\text{THIS OCCUR?}\)

Recall - choose 1st applicable rule.

\(\text{in DB, always choose}\)

append2([C], Y, Y) \(\Rightarrow\) append2([Y], Y, Z).

\(\text{NEVER choose termination rule}\)

append2([], Y, Y). \(\text{!!}\)
**RULE ORDER EFFECTS SOLDS**

- **RECALL** `append` vs. `append2`
- **COMPUTE**
  \[ \text{append}(x, [c], z) \text{ vs. append2}(x, [c], z) \]

```
prefix(L, [a, b, c]), suffix([a], L)
append(L, [a], [a, b, c]), suffix([a], L)
  \{ L -> L + [a], z \}
    suffix([a], [1])
      append(L, [a], [1])
        backtrack
          append(L, [a], [a, b, c]), suffix([a], [a, b, c])
            \{ L -> L + [a, c] \}
              suffix([a], [1])
                append(L, [a], [1])
                  \{ L -> L + [a, c] \}
                    suffix([a], [1])
                      append(L, [a], [1])
                        \{ L -> L + [a, c] \}
                          \{ \text{yes} \}
```
Cuts in Prolog

- Used to probe search tree of possibilities by eliminating choices when backtracking occurs.

- Why is this important?

1. Program execution improves since "you" inform the computer about goals which never contribute to a solution.

2. Program memory requirements are reduced due to smaller trees & less backtracking.

- Let's see some examples

Example: Library application

into oo - books
- who has borrowed what
- who to books are due back


client ('A. Jones').
client ('C. Water').

facility (Pat, Fee) :- book_overdue (Pat, Book), basic_fee (Pat, Fee),
     general_fee (Fee). \[ T \]
     basic_fee (reference).
     basic_fee (enquiries).
     general_fee (Pat).
     additional_fee (borrowing).
     additional_fee (interlibrary loan).

Using this example, how do cuts work?
For all clients, find out the facilities open to them.
?
\( \text{client}(x), \text{facility}(x, y). \)

**EXECUTING SNAPSHOTS**

- **client('A. Jones')**
- **facility('A. Jones', y)**
- **book.owned('A. Jones', book2407)**
- **! CUTOFF EXHAUSTED**
- **book.allocated(y)**

- **CUT IS SIMILAR TO A COMMIT - SETS AN UNPASSABLE BREAKPOINT**
- **ALL CHOICES MADE SO FAR ARE COMMITTED**
- **ONCE FIRST BOOK IS FOUND, NO OTHERS ARE SEARCHED FOR, FOR A. JONES.**
- **BACKTRACKING GOES TO CLIENT**

**WHY IS THE CUTOFF USED HERE?**

- **WHAT DOES IT MEAN?**

  - **THE QUERY WAS ONLY CONCERNED WITH WHETHER THE CLIENT HAD A OVERDUE BOOK, NOT ALL OVERDUE BOOKS FOR THE CLIENT.**

**HAS THE SEARCH TREE BEEN PRUNED?**

- **YES! - AFTER BACKTRACK HAVE BEEN DETERMINED, BACKTRACKING RESPECTS TO THE CLIENT RULE, BY PASSING BOTH THE BOOK.OWNED AND FACILITY ROLES. THUS, THE TREE HAS BEEN RADICALLY CUT.